Seminář Java

IV

Rekapitulace

- Deklarace tříd
  - Proměnné, metody, konstruktory, modifikátory přístupu
- Datové typy
  - primitivní, objektové, pole
- Dědičnost
- Rozhraní
- Řídící konstrukce
  - Podmínky, cykly
- Tvorba dokumentace

- news://news.fee.vutbr.cz/vutbr.fit.courses.ija

Hierarchie dědičnosti

- Třída *Object* je předkem všech tříd.
- Definuje základní množinu operací
  - public boolean equals (Object obj);
  - public int hashCode ( );
- Do proměnné, jejíž typ je deklarován jako třída A, lze dosadit všechny objekty třídy A a jejich podtříd.

Operátory a výrazy, porovnávání objektů

- Aritmetické
- Logické
- Relační
- Bitové
- Operátor podmíněného výrazu ? :
- Operátory typové konverze (přetypování)
- Operátor zřetězení +
- Porovnávání objektů
- Priority operátorů a vytváření výrazů

Aritmetické

- +, -, *, / a % (zbytek po celočíselném dělení)
- platí podobná pravidla jako v C/C++
  - `int / int` ⇒ `int`
  - `double / int` ⇒ `double`
  - `short / int` ⇒ `int`

Logické

- logické součiny (AND):
  - & (nepodmíněný - vždy se vyhodnotí oba operandy),
  - && (podmíněný - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)
- logické součty (OR):
  - | (nepodmíněný - vždy se vyhodnotí oba operandy),
  - || (podmíněný - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)
- negace (NOT):
  - !

Bitové

Bitové:

- součin &
- součet |
- exkluzivní součet (XOR) ^ (znak "stříška")
- negace (bitwise-NOT) ~ (znak "tilda")

Posuny:

- vlevo << o stanovený počet bitů
- vpravo >> o stanovený počet bitů s respektováním znaménka
- vpravo >>> o stanovený počet bitů bez respektování znaménka

Operátor podmíněného výrazu ? :

Bitové:

- Jediný ternární operátor
- Platí-li první operand (má hodnotu true) $\Rightarrow$
  - výsledkem je hodnota druhého operandu
  - jinak je výsledkem hodnota třetího operandu
- Typ prvního operandu musí být boolean, typy druhého a třetího musí být přiřaditelné do výsledku.

```
if (a > b)
     c = a - b;
else
     c = b - a;
```

```
c = (a > b ? a - b : b - a);
```

Operátor zřetězení +

- Výsledkem je vždy řetězec, ale argumenty mohou být i jiných typů, např.
- sekvence int i = 1; System.out.println("promenna i=" + i); je v pořádku
- s řetězcovou konstantou se spojí řetězcová podoba dalších argumentů (např. čísla).
- Pokud je argumentem zřetězení odkaz na objekt o ⇒
  - je-li o == null ⇒ použije se řetězec null
  - je-li o != null ⇒ použije se hodnota vrácená metodou o.toString() (tu lze překrýt a dosáhnout tak očekávaného řetězcového výstupu)

Relační (porovnávací)

- Tyto lze použít na porovnávání primitivních hodnot:
  - <, <=, >=, >
- Test na rovnost/nerovnost lze použít na porovnávání primitivních hodnot i objektů:
  - ==, !=
  - pozor na porovnávání objektů: == vrací true jen při rovnosti odkazů, tj. jsou-li objekty identické. Rovnost obsahu (tedy "rovnocennost") objektů se zjišťuje voláním metody o1.equals(Object o2)
  - pozor na srovnávání floating-points čísel na rovnost: je třeba počítat s chybami zaokrouhlení; místo porovnání na přesnou rovnost raději používejme jistou toleranci: abs(expected-actual) < delta

Operátory typové konverze (přetypování)

- Podobně jako v C/C++
- Píše se (typ) hodnota
- např. (Ucet)o, kde o byla proměnná deklarovaná jako Object.
- Pro objektové typy se ve skutečnosti nejedná o žádnou konverzi spojenou se změnou obsahu objektu, nýbrž pouze o potvrzení, že běhový typ objektu je požadovaného typu - např. (viz výše) že o je typu Ucet.
- Naproti tomu u primitivních typů se jedná o úpravu hodnoty - např. int přetypujeme na short a "ořeže" se tím rozsah.

Porovnávání objektů

Použití ==

- Porovnáme-li dva objekty (tzn. odkazy na objekty) prostřednictvím operátoru == dostaneme rovnost jen v případě, jedná-li se o dva odkazy na tentýž objekt - tj. dva totožné objekty.
- Jedná-li se o dva obsahově stejné objekty existující samostatně, pak == vrátí false.

Chceme-li chápat rovnost objektů podle obsahu, tj.

- dva objekty jsou rovné (rovnocenné, nikoli totožné), mají-li stejný obsah, pak
- musíme pro danou třídu překrýt metodu equals, která musí vrátit true, právě když se obsah výchozího a srovnávaného objektu rovná.
- Nepřekryjeme-li equals, funguje původní equals přísným způsobem, tj. rovné si budou jen totožné objekty.

Porovnávání objektů – příklad

Dva objekty třídy Ucet jsou shodné, mají-li stejného majitele a zůstatek.

```
public class Ucet implements Comparable {
    String majitel;
    double zustatek;
    public Ucet (String jmeno) {
        majitel = jmeno;
    }
    public boolean equals(Object o) {
        if (o instanceof Ucet) {
            Ucet c = (Ucet)o;
            return (zustatek == c.zustatek ?
                 majitel.equals(c.majitel) : false);
        } else
            return false;
    }
}
```

Metoda hashCode

Jakmile u třídy překryjeme metodu equals, měli bychom současně překrýt i metodu hashCode:

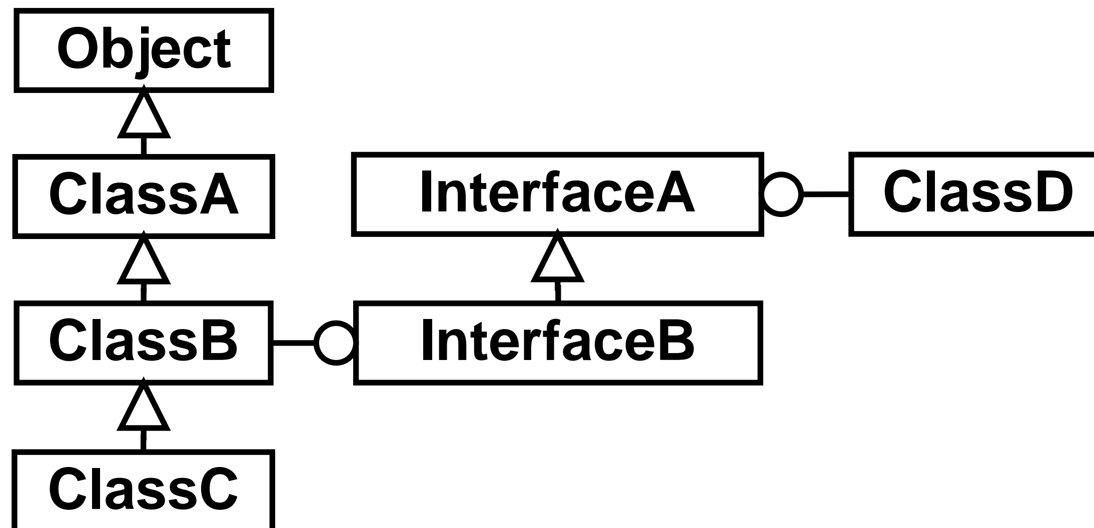
- hashCode vrací celé číslo (int) "co nejlépe" charakterizující obsah objektu, tj.
- pro dva stejné (equals) objekty musí vždy vrátit stejnou hodnotu.
- Pro dva obsahově různé objekty by hashCode naopak měl vracet různé hodnoty (ale není to stoprocentně nezbytné a ani nemůže být vždy splněno). Metoda hashCode totiž nemůže vždy být prostá.

Metoda hashCode - příklad

V těle hashCode často delegujeme řešení na volání hashCode jednotlivých složek objektu - a to těch, které figurují v equals:

```
public class Clovek implements Comparable {
    String majitel;
    double zustatek;
    public Ucet (String jmeno) {
        majitel = jmeno;
    }
    public boolean equals(Object o) {
        ...
    }
    public int hashCode() {
         return prijmeni.hashCode();
    }
}
```

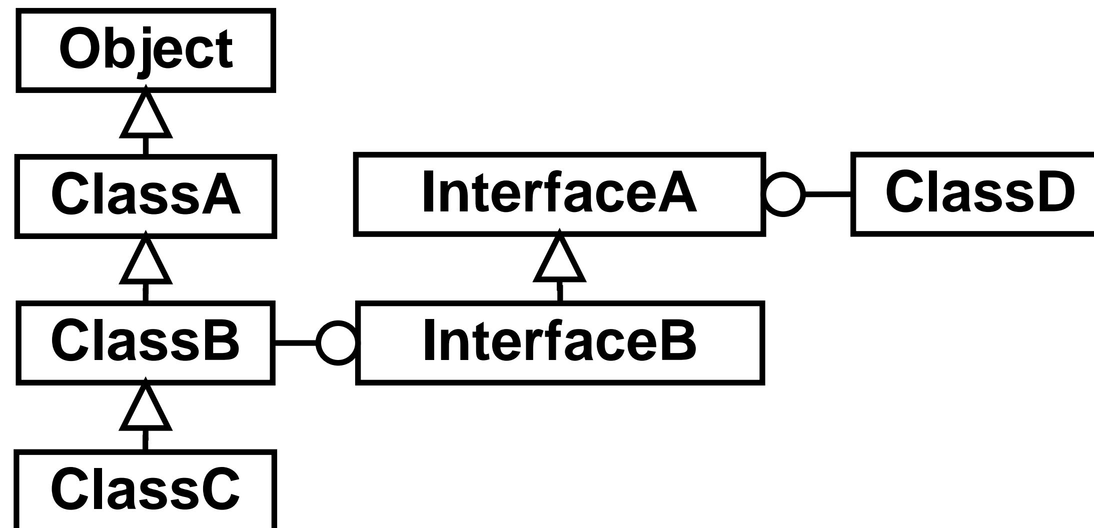
Dosazení objektu do proměnné – I




```
void method(ClassA o) { ... }
```

- o ⇒ ClassA, ClassB, ClassC
- o == ClassB ⇒ (ClassB) o

Dosazení objektu do proměnné – II




```
void method(InterfaceB o) { ... }
```

- o ⇒ ClassB, ClassC

```
void method(InterfaceA o) { ... }
```

- o ⇒ ClassB, ClassC, ClassD
- o == ClassC ⇒ (InterfaceB) o
- o == ClassC ⇒ (ClassC) o

Dosazení objektu do proměnné – příklad

```
class A {
    int i = 10;
    public int value() { return i; }
}

class B {
    int i = 20;
    public int value() { return i; }
}
```

```
    public void method1() {
        ...
        method2(new B());
    }
    public void method2(Object o) {
        A a = (A) o;    ⇐ chyba (ClassCastException)
        System.out.println(a.value());
    }
```

Modifikátor final

- Deklaruje konečný (neměnný) stav
- Třídy
  - `public final class Ucet { ... }`
  - od této třídy nelze "dědit" (vytvářet její potomky)
- Metody
  - `public final void print() { ... }`
  - tato metoda nemůže být "překryta" (overloaded) v odvozených třídách (potomci)
- Proměnné
  - `protected final int i = 10;`
  - `protected final String s = "řetězec";`
  - `protected final Banka b = new Banka();`
  - obsah proměnné je neměnný
  - konstanta

Výjimky

Co a k čemu jsou výjimky

- podobně jako v C/C++, Delphi
- výjimky jsou mechanizmem, jak psát robustní, spolehlivé programy odolné proti chybám "okolí" - uživatele, systému...
- není dobré výjimkami "pokrývat" chyby programu samotného - to je hrubé zneužití
- režie spojená s vyvoláním výjimky je vysoká

Výjimky

- Výjimka (Exception) je objekt třídy java.lang.Exception
- Objekty - výjimky - jsou vytvářeny (vyvolávány) buďto
  - automaticky běhovým systémem Javy, nastane-li nějaká běhová chyba, např. dělení nulou, nebo
  - jsou vytvořeny samotným programem, zdetekuje-li nějaký chybový stav, na nějž je třeba reagovat - např. do metody je předán špatný argument
- Vzniklý objekt výjimky je předán buďto:
  - v rámci metody, kde výjimka vznikla - do bloku catch $\Rightarrow$ výjimka je v bloku catch tzv. zachycena
  - výjimka "propadne" do nadřazené (volající) metody, kde je buďto v bloku catch zachycena nebo opět propadne atd.
- Výjimka tedy "putuje programem" tak dlouho, než je zachycena
- $\Rightarrow$ pokud není, běh JVM skončí s hlášením o výjimce

Syntaxe kódu s ošetřením výjimek

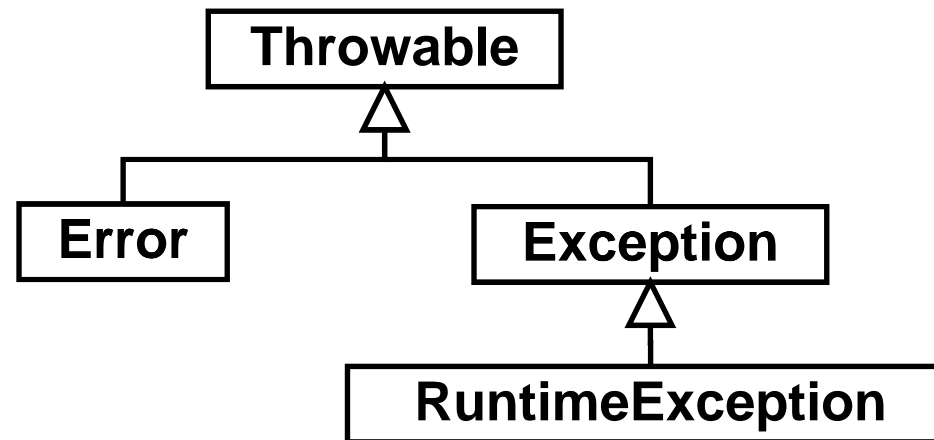
Základní syntaxe:

```
try {
        //zde může vzniknout výjimka
} catch (TypVýjimky proměnnáVýjimky) {
        // zde je výjimka ošetřena
        // je možné zde přistupovat k proměnnéVýjimky
}
```

- Bloku try se říká hlídaný blok, protože výjimky (příslušného hlídaného typu) zde vzniklé jsou zachyceny.
- V bloku catch jsou zachycené výjimky ošetřeny

Hierarchie výjimek

package java.lang



- Throwable – pouze objekty této třídy (a podtříd) mohou být generovány jako výjimky
- Error – vážné chyby JVM (*Out Of Memory, Stack Overflow, ...*)
- Exception – hlídané výjimky (checked exceptions)
- RuntimeException – běhové (runtime, nehlídané – unchecked) výjimky, takové výjimky nemusejí být zachytávány

Hlídané výjimky

```
package IJA.seminar4;
import java.io.*;
public class OtevreniSouboru {
    public static void main(String[] args) {
        String jmeno = args[0];
        FileReader r;
        System.err.println("Otviram soubor "+jmeno);
        r = new FileReader(jmeno);
        System.err.println("Soubor otevren");
        r.close();
    }
}
```

```
unreported exception java.io.FileNotFoundException;
must be caught or declared to be thrown
         r = new FileReader(jmeno);
unreported exception java.io.IOException; must ...
         r.close();
```

Ošetření výjimky

```
public static void main(String[] args) {
    String jmeno = args[0];
    FileReader r;
    System.err.println("Otviram soubor "+jmeno);
    try {
        r = new FileReader(jmeno);
        System.err.println("Soubor otevren");
        r.close();
    } catch (IOException ex) {
        System.err.println("Chyba pri manipulaci se
                                    souborem.");
    }
}
```

Ošetření výjimky

```
public static void main(String[] args) {
    String jmeno = args[0];
    FileReader r;
    System.err.println("Otviram soubor "+jmeno);
    try {
        r = new FileReader(jmeno);
        System.err.println("Soubor otevren");
        r.close();
    } catch (FileNotFoundException ex) {
        System.err.println("Soubor nelze otevrit.");
    } catch (IOException ex) {
        System.err.println("Chyba pri manipulaci se
                                    souborem.");
    }
}
```

Výjimka, které lze předejít

```
Exception in thread "main"
        java.lang.ArrayIndexOutOfBoundsException: 0 ...
```

```
public static void main(String[] args) {
    if (args.length == 0) {
        System.err.println("Nebylo zadano jmeno souboru");
        return;
    }
    ...
}
```

Propuštění výjimky

public FileReader(String fileName) throws FileNotFoundException;
public void close() throws IOException;

```
public class OtevreniSouboru {
    static void otevri(String jmeno) {
        System.err.println("Otviram soubor "+jmeno);
        FileReader r = new FileReader(jmeno);
        r.close();
    }
    public static void main(String[] args) {
        otevri(args[0]);
        System.err.println("Soubor otevren");
    }
}
```

Propuštění výjimky

```
public class OtevreniSouboru {
    static void otevri(String jmeno)
    throws IOException {
        System.err.println("Otviram soubor "+jmeno);
        FileReader r = new FileReader(jmeno);
        r.close();
    }
    public static void main(String[] args) {
        try {
            otevri(args[0]);
            System.err.println("Soubor otevren");
        } catch (IOException ioe) {
            System.err.println("Nelze otevrit soubor");
        }
    }
}
```

Reakce na výjimku

Jak můžeme reagovat?

- Napravit příčiny vzniku chybového stavu - např. znovu nechat načíst vstup
- Poskytnout za chybný vstup náhradu - např. implicitní hodnotu
- Operaci neprovést ("vzdát") a sdělit chybu výše tím, že výjimku "propustíme" z metody

Výjimková pravidla:

- Vždy nějak reagujme! Neignorujme, nepotlačujme, tj.
- blok catch nenechávejme prázdný, přinejmenším vypišme e.printStackTrace()
- Nelze-li reagovat na místě, propusťme výjimku výše (a popišme to v dokumentaci...)

Klauzule finally

Klauzule (blok) finally:

- Může následovat ihned po bloku try nebo až po blocích catch
- Slouží k "úklidu v každém případě", tj.
  - když je výjimka zachycena blokem catch
  - i když je výjimka propuštěna do volající metody
- Používá se typicky pro uvolnění systémových zdrojů - uzavření souborů ...

Vlastní výjimky

- Typy (=třídy) výjimek si můžeme definovat sami
- bývá zvykem končit názvy tříd - výjimek - na Exception

```
class MyException extends Exception {
     protected int pocetParametru;
     public MyException(int pocet) {
          pocetParametru = pocet;
     }
     public int getPocetParametru() {
          return pocetParametru;
     }
}
```

Ukázka vlastní výjimky a klauzule finally

```
public static void main(String[] args) {
    int pocetParametru = args.length;
    try {
        if (pocetParametru < 2)
            throw new MyException(pocetParametru);
        System.out.println("Spravny pocet parametru: "
                           +pocetParametru);

    } catch (MyException mp) {
        System.out.println("Malo parametru: "
                           +mp.getPocetParametru());

    } finally {
        System.out.println("Konec");
    }
}
```

Ladění programu - I

- kontrolní tisky - System.err.println(...)
- řádkovým debuggerem jdb
- integrovaným debuggerem v IDE
- pomocí speciálních nástrojů na záznam běhu balíků

- používat systémy pro běhovou kontrolu platnosti podmínek:
  - vstupní podmínka metody (zda je volána s přípustnými parametry)
  - výstupní podmínka metody (zda jsou dosažené výstupy správné)
  - a podmínka kdekoli jinde - např. invariant cyklu ...

Ladění programu - II

- standardní klíčové slovo (od JDK1.4) assert booleovský výraz
- testovací nástroje typu JUnit (a varianty - HttpUnit,...) - s metodami assertEquals() apod.
- pokročilé nástroje na běhovou kontrolu platnosti invariantů, vstupních, výstupních a dalších podmínek - např. jass (Java with ASSertions), http://csd.informatik.uni-oldenburg.de/~jass/

Ladění programu - assert

```
public class Zlomek {
      int cit, jm;
      public Zlomek(int c, j) {
            assert j != 0;
            cit = c;
            jm = j;
      }
}
```

- Přeložit jej s volbou -source 1.4
- Spustit jej s volbou -ea (-enableassertions)
- Dojde-li za běhu programu k porušení podmínky stanovené za assert, vznikne běhová chyba (AssertionError) a program skončí.

Souhrn

- Deklarace tříd
  - Proměnné, metody, konstruktory, modifikátory přístupu
- Datové typy
  - primitivní, objektové, pole
- Řídící konstrukce
  - Podmínky, cykly
- Dědičnost
- Rozhraní
- Porovnávání objektů
- Výjimky
- Testování
- Tvorba dokumentace

Klíčová slova – I

Data Building Blocks (Built-In Types)	Statement Building Blocks
byte, short, int, long	break, continue
boolean	do, while
char	for
float, double	if, else
null	switch, case, default
	throw
	try, catch, finally

Klíčová slova – II

Class Stuff	Class/Method Stuff
class	abstract
extends	private, public, protected
implements	final
instanceof	static
interface	throws
new	
super, this	

Method Stuff	Larger-Than-Class Building Blocks
return	import
void	package